Break the Windows: Explicit State Management for Stream Processing Systems

Margara, Alessandro ; Dell'Aglio, Daniele ; Bernstein, Abraham

DOI: https://doi.org/10.5441/002/edbt.2017.50

Poster Paper

Break the Windows: Explicit State Management for Stream Processing Systems

Alessandro Margara
DEIB, Politecnico di Milano
alessandro.margara@polimi.it

Daniele Dell'Aglio
IFI, University of Zurich
dellaglio@ifi.uzh.ch

Abraham Bernstein
IFI, University of Zurich
bernstein@ifi.uzh.ch

ABSTRACT

Several stream processing and reasoning systems have emerged in the last decade, motivated by the need to process large volumes of data on the fly, as they are generated, to timely extract relevant knowledge. Despite their differences, all these systems isolate the data that is relevant for processing using (fixed size) windows that typically capture the most recent data and assume its validity.

We claim that this paradigm is not flexible enough to effectively model several application domains and we propose a novel abstraction that enables for explicit state representation and management. We model state as a collection of data elements annotated with their time of validity and we augment the traditional stream processing paradigm with state-handling abstractions to declare how the input streams affect the state of the system and how the state influences the results of the processing.

Keywords

Stream Processing; Stream Reasoning; Event Processing; Windows; State Management; Explicit State

1. INTRODUCTION

Several application domains require analyzing *streams* of data on-the-fly, as new data become available, to extract valuable knowledge. Examples include environmental monitoring, click stream analysis in Web sites, traffic monitoring, credit card fraud detection, computer systems monitoring, interaction analysis in social media, and smart cities.

In the last decade, this need led to a bloom of technologies for *stream processing* and *reasoning* [6, 12], which introduce (i) languages and programming abstractions to define how to extract relevant knowledge from the input data; (ii) algorithms and techniques to efficiently perform such task.

Stream processing and reasoning systems were developed by researchers and practitioners active in diverse fields, such as database systems [3], event-based systems [11], knowledge representation [5], and Big Data processing [13]. As a consequence, they present heterogeneous design choices and characteristics.

Despite their differences, all these technologies build on the implicit assumption that the most recent data is also the most relevant for processing, and isolate recent data using *window* operators, for instance to limit the scope of the analysis to the last ten elements of the stream or to the elements that occurred within the last five minutes. Furthermore, they often assume that *all* the information stored in windows is valid when the processing takes place.

We believe that this paradigm is not flexible enough to effectively model several application domains, since: (i) windows with a predefined and fixed size might not be suitable to define the portion of streaming data that is relevant for processing, which might depend on the specific content of the data; (ii) windows might include invalid or contradictory information.

To motivate our claims, let us refer to some concrete use cases. Consider a click-stream monitoring system that analyzes the interactions of potential customers with an e-commerce Web site. The system should trace a user from the moment when she enters the Web site to the moment when she leaves the Web site. A shorter observation time frame would be meaningless for the application logic, whereas a larger time frame could waste computational resources. This simple example highlights that fixed-size windows are not always adequate to isolate relevant stream elements.

Consider now a security service to monitor the position of visitors in a building, in which sensors signal a new event every time a visitor enters a room. If we assume a fixed time-window of five minutes, it is possible that a visitor moves through multiple rooms within the scope of a single window. Considering all the events generated within this fixed time frame as valid would lead to the erroneous conclusion that the visitor is simultaneously in multiple rooms. This example shows that one might infer contradictory information if she simply considers as valid all the data elements within a window, without properly considering their mutual relations —in our example, the most recent position invalidates and updates any previous position of the same visitor—.

We believe that the above limitations could be overcome with flexible abstractions to model *state* in stream processing systems. For instance, the e-commerce scenario could benefit from the presence of state information that records which users are active at a given point in time. Similarly, the security service could model the position of each visitor as part of the state and update such a state whenever the visitors move.

Moving from these premises, we propose an extension to stream processing and reasoning that makes state explicit and captures it as a first class object. We model state as a collection of data elements annotated with their time of validity. Then, we augment the traditional stream processing model that defines transformations from input streams to output streams with state-handling abstractions to (i) declare how state information influences the results of the processing —for instance, we want to monitor only active users in the e-commerce scenario, where the set of active users is defined as part of the state—; (ii) declare how the stream of input data updates the state —for instance, a new event in the security service invalidates previous information about the position of a visitor and adds a new state element with the current position of that visitor—.

We see several benefits from explicit state management in stream processing systems. As motivated by the scenarios above, the possibility to define and reference state information has the potential to ease the modeling of the application at hand. Furthermore, it might simplify the processing task by activating some derivations only when specific conditions on the state are met. Finally, the presence of an explicit repository for state information would make the state available for query and retrieval.

The paper is organized as follows. Section 2 presents the state-of-the-art systems for stream processing and reasoning, with emphasis on their approaches to manage state information. Section 3 presents the model we propose to explicitly manage state. Section 4 surveys work that is related to the topic of this paper, and Section 5 concludes the paper and draws a road map for future work.

2. BACKGROUND

This section surveys the main models and technologies for stream processing and reasoning, focusing on their state management capabilities.

Perhaps the first processing model for streaming data is CQL —Continuous Query Language— that builds on the relational model [3]. CQL introduces *window* operators to isolate finite blocks of the input streams and then applies relational processing on such blocks. Windows typically include the latest elements received from the input streams: as new data is received, the content of the windows changes, and the processing is re-executed to update the results accordingly. The most widely adopted windows have a fixed size in terms of number of elements —count windows— or time span —time windows—. This model is the core of virtually all Data Stream Processing Systems (DSMSs) [4].

The relational core of this model facilitates the interoperability of streaming data and static relational tables. Although static tables could be theoretically employed to store state information, the model does not include state-management functionalities.

Complex Event Processing (CEP) systems consider each stream element as the notification of occurrence of an *event* at some points in time, and offer abstractions to define situations of interest as (temporal) patterns of events [6, 11]. Patterns are typically searched for within time windows or include temporal constraints conceptually similar to time windows.

Some CEP systems adopt interval time semantics, meaning that the situations detected from the raw events can have an associated time *interval* of validity [2]. Situations con encode the current state of the application environment and be composed with further events during processing. Nevertheless, these systems do not offer specific abstractions to model and update state information. Furthermore, situations are not persisted and cannot be queried.

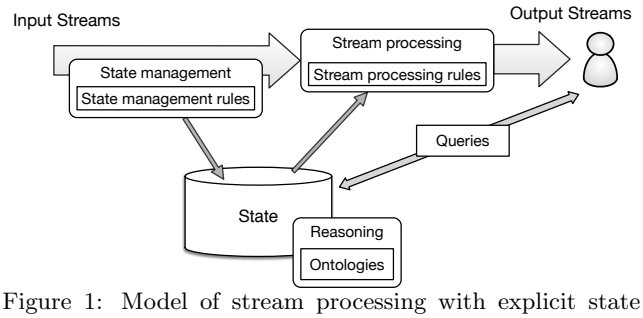

Figure 1: Model of stream processing with explicit state management

Stream reasoning extends the above approaches by exploiting the RDF data model to represent data elements within the streams, which enables for complex forms of logic inference —*reasoning*— [12], often trading off performance for expressivity. Despite the use of a different data model and more expressive processing, stream reasoning systems inherit the windowing mechanisms of DSMSs and CEP systems. Furthermore, since they typically exploit *all* the information in the current window to perform logical inference, they might suffer from the presence of inconsistent data, as discussed in the use cases in Section 1.

Big Data processing systems were originally designed to batch process large volumes of data on large clusters of commodity machines. Nowadays, they are shifting from pure batch computation to streaming computations [14, 13]. These systems describe the computation as a directed graph of operators. Input data elements traverse this graph and get processed one by one or in small batches.

Similar to DSMSs, operators include windows to collect portions of the streaming data. Interaction with static databases is possible, but no abstraction is provided to model and update state information.

3. A MODEL FOR EXPLICIT STATE MANAGEMENT

Figure 1 presents the model we envisage to enable explicit state management in stream processing systems. The data received from the input streams is analyzed both in the `state management` component and in the `stream processing` component. The former elaborates the input data according to a set of deployed `state management rules` to update the current state of the system, stored in the `state` repository. The latter processes the input data —together with the `state` information— to continuously produce new results for the users according to a set of deployed continuous queries or, more in general, `stream processing rules`.

Users can also query the state of the system by submitting `queries` as in traditional database systems. Finally, a `reasoning` system can extract implicit knowledge from the explicit state information to augment the answers to both stream processing rules and one-time queries. Reasoning is based on a formal description of the application domain, in the form of `ontologies`.

3.1 Case study

We now exemplify the use of our model through a case study. Let us consider a decision support tool to manage an e-commerce Web site. The managers of the Web site want to receive constant updates about the current trends of product sales, the quality of service the Web site offers —for instance, the average delay to deliver the products—, and the status of the inventory.

Traditional stream processing systems can easily satisfy these needs by periodically computing aggregates over sliding windows. For instance, the current trend of sales could be computed by summing up all the sales for each class of products over a time window selected by the user.

Nevertheless, the products and their classification change over time: new products are constantly added, new classes are created, and previous classes of products are split or merged. The information about the products and their classification is managed by a different division of the company, which updates the management whenever it is needed.

The set of available products and their classification represent background state information that the management needs to consider to correctly interpret the trends of sales.

Several approaches are possible to capture this state information. On one extreme, state can be stored in a separate database that is updated manually or semi-automatically whenever new information about the products becomes available. The stream processing system then accesses the current data in the database during processing. On the other extreme, the stream processing system might be responsible to process both the information about the sales *and* the information about the products and their classification. This approach complicates the stream processing rules, since they need to take into account heterogeneous types of data —sales and products classification— and their interaction. Furthermore, it becomes impossible to express all the processing by means of computations over sliding windows. Indeed, the system must ensure that *all* the information that builds up the most recent classification of products is taken into account, independently from the time when such information was generated.

In our model, we propose to explicitly store state information and enable the stream processing system to access that information during processing. We propose to encode the logic that updates the state based on the input streaming data into `state management rules` that the `state management` component uses to automatically handle state changes. This approach relieves the stream processing system from analyzing information related to the products and their classification, thus simplifying the `stream processing rules` that compute the selling trends.

Making the state explicit also enables the users to query such state, which would not be possible if the state information was only processed within the stream processing system. We envision the possibility to implement the `state` component as a temporal database, thus enabling the query and retrieval of both the current state and historical data. In our e-commerce case study, this enables the management to retrieve and analyze past information about products and sales to confront them with the current trends.

Finally, the `state` component can exploit domain information —for instance in the form of `ontologies`— to derive new knowledge from the explicit information it stores. For instance, in the e-commerce example, the ontology might include a taxonomy to organize the products according to different classification criteria and to automatically derive sub-classes relations.

3.2 The benefits of explicit state management

Based on the case study above, this section summarizes the benefits we see in our proposed approach.

Separation of concerns. The proposed approach decouples the management of state updates from the stream processing logic. The former is encoded in the `state management rules` while the latter is encoded in the `stream processing rules`. This enables the developers of the system to separately model these two orthogonal aspects.

Different abstractions. The separation of state management from the stream processing logic enables the adoption of separate abstractions for the two tasks. As discussed in Section 1 and in Section 2, most stream processing languages and systems are designed to express and perform continuous computations over moving windows, and this is not suitable to express state management tasks, since windows might miss some relevant state information or include contradictory data. By delegating the state management to separate rules, our approach can adopt different formalisms to express how the state is updated.

Queryable state. By making the state explicit, the proposed model enables the users to query the state on-demand, potentially referring to historical data. This would not be possible using only stream processing technologies that internally and implicitly store only the state required to execute the `stream processing rules`, and do not offer primitives to access such state. Also, queryable state can promote interoperability, since stream processing systems can expose their state and query the state of other systems.

Reuse of consolidated technologies. By clearly separating state management from stream processing, the proposed model can take advantage of consolidated technologies that are optimized for these purposes. For instance, the `stream processing` component can be easily implemented using state-of-the-art stream processing languages and systems, as presented in Section 2. Similarly, the `state` component can adopt well studied algorithms and technologies to optimize the evaluation of queries —coming both from the users and from the `stream processing` component— and to perform `reasoning` tasks.

3.3 Open research questions

This section highlights the open research questions that we are currently investigating to concretely implement the above model.

State management rules. The language used to express `state management rules` greatly influences the expressivity of the system. In the simplest case, state transitions are determined by some individual elements in the input stream. For instance, in our e-commerce use case, an individual input element might represent the new classification for a product. However, we envision more complex situations in which a state transition is determined by multiple streaming elements. We are currently investigating possible abstractions to capture these scenarios.

State representation, query, and retrieval. An open research question involves which state information to store —only the current state or also historical data—, how to represent

this information —for instance, using a relational database or a key-value store—, and which language to offer for state query and retrieval.

Interaction between stream processing and state. Perhaps the most challenging question is how to define the overall semantics of the system, taking into account the possible interactions between the state —and the `state management rules`— and the `stream processing rules`. Considering the e-commerce use case, we need to define how a change in the classification of products might impact on the ongoing streaming computation.

4. RELATED WORK

The limitations of fixed count or time windows in stream processing is well known in the literature. To overcome these limitations, some approaches propose windows that are based on the content of input elements. Li et al. [10] use content-based windows to define an effective evaluation strategy for window aggregates. Similarly, predicate windows [8] define views and support view maintenance in data stream processing systems. They predicate on the content of an input element to determine whether it has to be considered as new information, or as an update (or deletion) of existing information for a given view. Frames [9] provide the developers with built-in functions to simplify the statistical analysis of data. Google Dataflow [1] proposes the concept of *session windows*, which partition a stream based on some user defined field —for instance, the identifier of a session in an e-commerce Web site—. Our model builds on similar ideas and takes a step forward by enabling the developer to express state in a more general ways, using rules that define how input elements impact on state.

Perhaps the closest work to our proposal is TEF-SPARQL [7], an extension to the SPARQL query language that distinguishes events from *facts*, where the latter are similar to the timed data elements that build the state in our model. TEF-SPARQL provides ad-hoc operators to combine facts and events and a *replace* primitive to update the set of facts. Nevertheless, TEF-SPARQL encodes the whole logic to manage and update facts within stream processing rules, whereas we separate the inference of new knowledge —including new state elements— from stream processing rules. Furthermore, we enable on-demand query of state.

5. CONCLUSIONS

Virtually all the state-of-the-art stream processing and reasoning systems rely on fixed-size windows to isolate the portions of input streams that are relevant for processing. We move from the observation that this schema is not flexible enough to effectively model several application domains and we propose a novel approach that enables the users of a stream processing system to explicitly define and modify the state of the application scenario at hand.

We believe that the approach we propose can advance the state-of-the-art in stream processing and reasoning in two orthogonal ways: on the one hand, it can ease the modeling of application scenarios in which the state of the system plays a fundamental role; on the other hand, it can simplify the processing effort by limiting the amount of streaming data that needs to be analyzed depending on the specific state of the system.

In the near future, we plan to provide a more detailed and precise formalization of our model, to implement the model into a prototype stream processing system, and to evaluate the befits of the proposed approach in terms of modeling and processing. We will consider various real world use cases with different requirements in terms of expressivity and processing complexity.

6. REFERENCES

[1] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *PVLDB*, 8(12):1792–1803, 2015.

[2] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, pages 635–644. ACM, 2011.

[3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.

[5] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. Querying RDF streams with C-SPARQL. *SIGMOD Record*, 39(1):20–26, 2010.

[6] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3):15:1–15:62, 2012.

[7] S. Gao, T. Scharrenbach, J. Kietz, and A. Bernstein. Running out of bindings? integrating facts and events in linked data stream processing. In *SSN-TC/OrdRing@ISWC*, volume 1488 of *CEUR-WS Proceedings*, pages 63–74. CEUR-WS.org, 2015.

[8] T. M. Ghanem, A. K. Elmagarmid, P. Larson, and W. G. Aref. Supporting views in data stream management systems. *ACM Trans. Database Syst.*, 35(1), 2010.

[9] M. Grossniklaus, D. Maier, J. Miller, S. Moorthy, and K. Tufte. Frames: data-driven windows. In *DEBS*, pages 13–24. ACM, 2016.

[10] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD Conference*, pages 311–322. ACM, 2005.

[11] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2001.

[12] A. Margara, J. Urbani, F. van Harmelen, and H. E. Bal. Streaming the web: Reasoning over dynamic data. *J. Web Sem.*, 25:24–44, 2014.

[13] N. Marz and J. Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[14] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *SOSP*, pages 423–438. ACM, 2013.